# MIPS R4400MC Errata, Processor Revision 2.0 & 3.0

January 24, 1995

MIPS Technologies Inc.
2011 N Shoreline Blvd
PO Box 7311
Mountain View, CA 94039-7311

This document contains information that is proprietary to MIPS Technologies, Inc.

MIPS Technologies, Inc. reserves the right to change any products described herein to improve the function or design. MIPS Technolgies, Inc. does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under patent rights nor imply the rights of others.

Copyright 1993 by MIPS Technologies, Inc. All rights reserved. No part of this document may copied by any means without written permission from MIPS Technologies, Inc.

Additional errata which affect uniprocessor designs and may affect multiprocessor configurations are listed in the MIPS R4400 PC, R4400 SC errata. Change bars in the left column indicate corrections or changes from the last version of the errata.

1. Under the conditions listed below, the EB bit in the CacheErr register is incorrectly set.

1) A store targets a shared line in the primary cache
2) The tag in this line has a parity error
3) Under this condition, the processor will stall due to a data cache miss and the CacheErr register is set.
4) As the processor comes out of the data cache miss and before it vectors to the CacheErr exception vector, there is an instruction cache miss and a pending external request which targets the same line with the parity error.

Under these conditions, the EB bit will get set although there was no parity error. The EB bit implies that both a data and instruction parity error have occurred. In this case, there was only a data parity error.

Workaround: The EB bit is meaningful only if the ER bit in the CacheErr register indicates an instruction error.

2 When Create-Dirty-Exclusive-SD cacheop is performed on a line which is present in the processor in Shared or Dirty Shared state; the processor invalidates the line before modifying it to Dirty Exclusive state. This might create the following problem: If there is a snoop or an intervention to this line, while the processor is waiting for IvdAck, the processor could send an incorrect response with an Invalid state instead of Shared or DShared state.

Workaround:There is no workaround for this problem.

3. External Updates to a line which exists both in the PICache and PDCache at the same time causes the copy of the line in the SCache and PDCache to be updated but doesn't not change the state of the copy in the PICache. If the line is in the SCache and the PICache, only, then the processor properly updates the SCache line and invalidates the PICache line; and if the line is in the SCache and the PDCache, only, then the line gets updated in both secondary and primary caches, as expected.

Workaround: Do not allow a line to exist in both PDCache and PICache if an update protocol is used or use "write invalidate" protocol for the instruction space.

4. In this following sequence:

| | |
|---|---|
| ddiv | (or ddivu or div or divu) |
| dsll32 | (or dsrl32, dsra32) |

if an MPT stall occurs, while the divide is slipping the cpu pipeline, then the following double shift would end up with an incorrect result.

Workaround: The compiler needs to avoid generating any sequence with divide followed by extended double shift.

5. The processor sends Read Request with incorrect value of the *"Link Address Retained"* bit. This

error occurs when the following sequence of events takes place:

1) ICache miss to a line replacing link address in scache.
2) ICache Read request is stopped by de-asserting RdRdyB
3) An external request comes during this time and R4400 has to regenerate the address and command.

When the processor regenerates the Read Request after responding to the external request, it compares the link address register with a different address than the instruction address that caused the miss. As a result, sometimes it incorrectly sets or resets the *"Link Address Retained"* bit.

The consequence of incorrectly setting the *"Link Address Retained"* bit are not of any concern since the external agent would snoop assuming the line exist in shared state; but the processor would provide the state as Invalid. However, the consequence of incorrectly not indicating the *"Link Address Retained"* is significant since the atomic functionality could be broken.

Workaround: The hardware solution is to either not use the RdRdyB signal or in the case when the RdRdyB is used, to latch the retained bit when it occurs with the first Read Request even though the request is not accepted.

6. When a TLB refill exception occurs on an instruction fetch, the value in the CP0 register BadVAddr might not match CP0 register EPC (or EPC+4 in case of a branch or jump with the delay slot as the first instruction of the next page.

Workaround:

In the first level TLB refill exception, use the TLB probe instruction to check if the virtual address which is in the BadVAddr register already exist in the TLB. If it is in the TLB, then eret (as the BadVAddr was incorrect), else go ahead and write the new TLB entry and eret. By overlapping the TLB probe operation with the other instructions in the handler, and then placing the TLB write instruction in the branch delay slot of a branch likely instruction, the performance overhead for this workaround can be minimized.

Example:

```
        mfc0    k0,   context
        lw      k1,   0(k0)
        lw      k0,   4(k0)
        c0      tlbp              <-- additional instruction due to workaround
        srl     k1,   k1, 3
        mtc0    k1,   tlblo0
        srl     k0,   k0, 3
        mfc0    k1,   index       <-- additional instruction due to workaround
        mtc0    k0,   tlblo1
        bltzl   k1,   1f          <-- additional instruction due to workaround
        c0      tlbwr
1:      nop
        c0      eret
```

If the processor takes a TLB refill exception from the first level exception then it will jump to the "general exception handler". Inside the "general exception handler", when the operating system (OS) detects an address outside the expected range in BadVAddr, it should check EPC to make sure it is within a valid range for the process. If EPC is within the valid range, the OS should execute an “eret” instruction. The refill instruction will be re-taken and BadVAddr will contain the correct value.

If the OS is unable to determine the valid address range for the process, the value in EPC should be used to look for a load or store instruction. If EPC does not point to a load or store, the OS should execute an “eret”. The “eret” will then cause another TLB refill exception, which will have a valid BadVAddr. If EPC points to a load or store, the OS must then interpret the instruction to generate the address for the data. If this address matches the address in BadVAddr, the process tried to access data outside the process address space. Otherwise the OS should execute an “eret” causing a TLB refill exception where the value in BadVAddr will be valid.